NoSQL Write-up – Dota2 Game Replay Analysis

Team name: PSG.LGD

Yichun Yan
Ziwei Jiang
Yifan Li
Weiqi Wang

November 1, 2019

Contents

1 Datasets **3**
1.1 Background of Datasets . . . 3
1.2 Source of Datasets . . . 4
1.3 Descriptions of Datasets . . . 4
1.4 Data Dictionaries . . . 4

2 Data ETL **5**
2.1 Extraction . . . 5
2.2 Transformation . . . 5
2.3 Loading . . . 6

3 Overview of our Architecture **7**

4 Data Analysis **7**
4.1 Bad Manner Hero . . . 8
4.2 Game Time . . . 8
4.3 Damage/Gold Rate . . . 8
4.4 Economy Distribution vs Game Result . . . 9
4.5 Gain in First 15 Minutes . . . 10
4.6 First Blood Time vs Game Time . . . 10
4.7 Hero Statistics . . . 10
4.8 Most Ban . . . 11
4.9 Most Pick . . . 11
4.10 Most Used/Purchased Item . . . 11
4.11 Team Battle Detector . . . 12
4.12 Win-lose Relationship Visualization . . . 12

5 Challenges **13**
5.1 Valve API . . . 13

5.2 Dota2 replays Parameters . . . 14
5.3 Docker . . . 14
5.4 Setup AWS environment . . . 14
5.5 Result Readability . . . 15
5.6 Corrupted Data . . . 15
5.7 Setup Neo4j on AWS . . . 15
5.8 Missing Output on Console . . . 15
5.9 Limited Information in Parsed Replay Data . . . 15

6 Appendix **15**
6.1 Detailed Data Dictionaries . . . 15
6.1.1 Match Result Dataset . . . 15
6.1.2 Parsed Replay Dataset . . . 18
6.2 Detailed Analysis Results . . . 19
6.2.1 Damage/Gold Rate . . . 19
6.2.2 Hero Statistics . . . 21
6.2.3 First 15 min Gold/XP Gain . . . 24
6.2.4 Most Pick Hero . . . 24
6.2.5 Most Purchased/Used Item . . . 25

1 Datasets

1.1 Background of Datasets

Our team aims at exploring data about a popular and long-lived computer game, Dota2.

Dota2 is a multiplayer online battle arena (MOBA) video game developed and published by Valve Corporation. Each game will involve ten players that are divided to two teams play against each other on a same map. Every player chooses one "hero" that has unique abilities and different styles of playing. The aim for each team is to destroy the building called "Ancient" of the other team. Players collect gold to buy powerful items and gain XP by killing players on the other team, destroy building of the other team, or kill creeps.

Figure 1: A screenshot of the game

This game has a large quantity of players, accordingly a huge amount of replays will be produced every day. Also, it's a real-time game, hundreds of operations and some decisions will be made each second, and all of them will have some influence on the final results of the game. Both of the above two features make the game a great subject to big data analysis.

Actually, there are some professional teams start hiring data engineers to help their players perform better in a game. This means that our analysis has some realistic meaning. For example, by using big data pipeline to answer the business question "Does there exist a regular farming path for some professional players?", we can understand both our teammates and the opponents better, and therefore optimize our strategies.

We have used technologies like MongoDB, Apache Spark, Neo4j for this project. All of the code of this project can be found in this repository, and I will explain the logic of our project and reference to some lines of code when necessary in the following sections.

1.2 Source of Datasets

Dataset Name	Source	Format
Dota2 match result dataset	Valve's official API	JSON file in this schema
Dota2 replay dataset	Valve's Dota2 replay servers [1]	`.dem` binary format, designed for Dota2 game client and need to be parsed

1.3 Descriptions of Datasets

1. Dota2 match result dataset: Snapshot of the end of a game, including the winner team, the duration of the whole game, each player's ending level, items and kill/death/assist, and other important information about the game and each player.
2. Dota2 replay dataset: Everything happens in a game, which includes the whole combat log of the game, chatting information and the spawn and death of NPCs.

We will discuss how we acquire those data how it is transformed into a usable format in details in the ETL selection.

1.4 Data Dictionaries

The match result data is already in JSON format, we can use it straightforwardly in the same schema to save it into our database. The data dictionary for this dataset can be found in table 2, 3 and 4 in the Appendix.

The replay dataset is initially binary files, therefore we have to parse it first then fit into a schema that we design. After these process, one document(record) of a replay should have the following fields:

Field Name	**Data Type**	**Description**	**Example**
match_id	long	The unique match id of this match	4986514901
combatlog	JSON Array	Describing all the events in the game. See table 5 for details	See table 5
info	JSON Object	Basic information about the match. See table 6 for details	See table 6
chat	JSON Array	What the players say in the chat during the match	`{ "sender": "Sk SpwaN", "message": "gg wp" }`
lifestate	JSON Array	Spawns and deaths of objects in the game	`{ "tick": 7420, "type": "spawn", "object": "CDOTA_BaseNPC_Creep_Lane" }`

Table 1: Replay Data Dictionary

[1] urls for downloading replay can be constructed by combining the information in an OpenDota API and one of the Valve's Official API.

2 Data ETL

Our aim is to collect about 1,500,000 records for our match result dataset and about 1000 professional games, 1000 ranked games and 1000 public games for our replay dataset. Our data is mainly three sources:

1. Valve's official api.
2. OpenDota's third party api.
3. Dota2's replay cluster.

We will walk through how we create a pipeline to make use of the above three sources to collect all the data we want in the Extraction section.

2.1 Extraction

In order to call the Valve's official api, we need a key to get authorized. Each key has its api call limits, so we create a key pool that contains 5 different authorizing keys that are obtained using our Steam accounts. Every time we need to call the api, we randomly pick one key from the poll.

The api we are mainly using is GetMatchHistoryBySequenceNum. It returns an array of game result information in its `matches` field[2]. It requires two parameters: `start_at_match_seq_num` and `matches_requested`. The first one is an unique identifier for each match, and the second one is the number of mathch histories to be returned. We construct the url for the api call to get a bunch of match results and save them into MongoDB. Also, since we always want to keep the HiFi data around, I also wrote the original JSON file into the disk.

After we get a list of match result in the previous call, we can use the sequence number of the last record we get from the last call then plus one as the `start_at_match_seq_num` for the next iteration.

In each API call, I iterate the whole results to find out whether there is a valid professional game. If a valid professional game is found, we will call OpenDota's API using the `match_id`, which is a part of the previous results, to obtain `replay_salt` and `cluster`, the information we need to download the replays.

We construct such url: `http://replay<cluster>.valve.net/570/<match_id>_<replay_salt>.dem.bz2` and start another thread to to download, unzip and parse the replay. At the same time, since we want to get a same number of ranked matches and public matches, we maintain two integers to record how many public matches and ranked matches to download. As we start downloading a professional match, we add one to those two integers, and if the numbers are not zero when we encounter a public match result or a ranked match, we start to download them using the same method I used for professional games. This logic can be mapped to this snippet of code.

In this way, we can get game replays of three types in same amount for the second dataset while we collect the data for the first dataset.

2.2 Transformation

The match result dataset is easy to handle, since the result of in JSON format, which can be easily parsed into Document datatype by the MongoDB Driver in Java and saved to MongoDB.

However, the replay is in binary format, which is designed to be executed by the Dota2's game client. Luckily, we found open-source parser, clarity, to help us transform it into something useful.

We create a pipeline to download, unzip each replay, parse it into Documents with the schema we mentioned in table 1, and finally store the parsed Document into MongoDB.

[2]The schema of each match result is the same as what described in GetMatchDetails API.

The code to parse the `.dem` file into our schema is as follows:

1. The overall logic.
2. For the "info" field.
3. For the "combatlog" field .
4. For the "chat" field.
5. For the "lifestate" field.

2.3 Loading

We deploy our database on AWS EC2. This snippet of code is used to save the JSON format match result data into MongoDB and this snippet is for writing parsed replay data into MongoDB.

The size of our dataset is:

Dataset Name	Number of records	Storage Size
Dota2 match result	About 1,500,000	About 15 GB
Dota2 replay dataset	About 3500	About 35 GB

As described in the proposal, we also applied Neo4j to support relationship visualization. We wrote Neo4jLoader to extract the data from MongoDB, filter it with the help of Spark and load it into our Neo4j database.

3 Overview of our Architecture

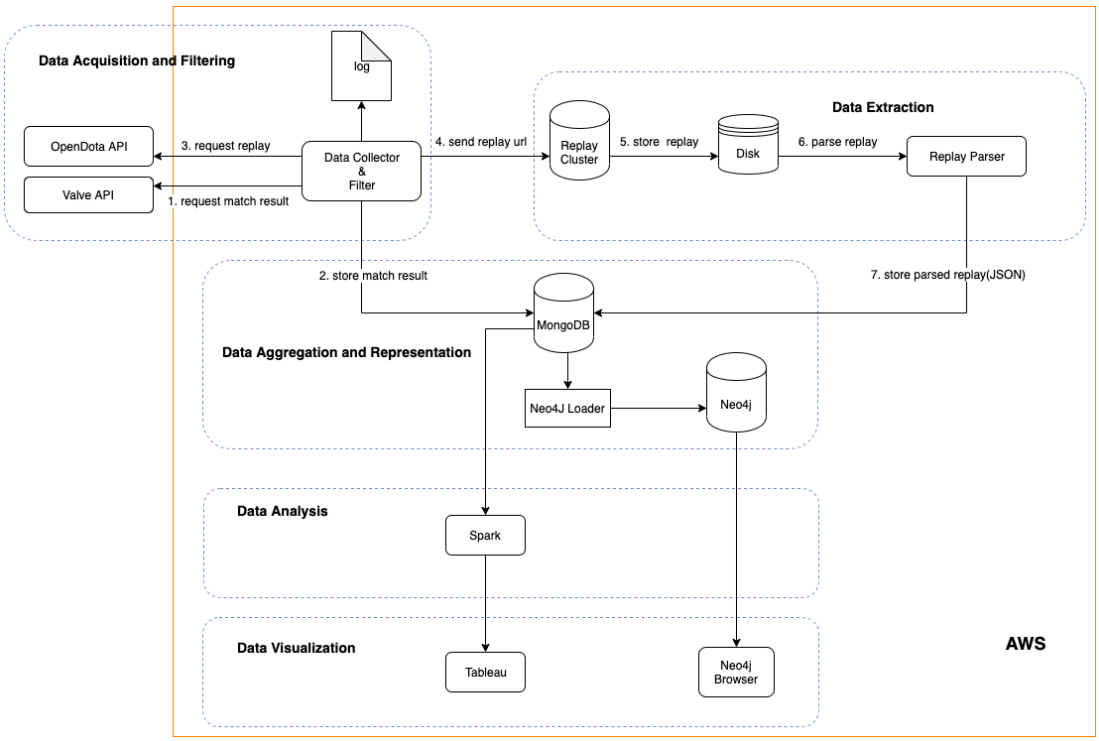

Figure 2: System Architecture

The whole pipeline starts from top-left, where the data collector send API requests to ValveAPI and OpendotaAPI. Log is output to the disk so that we know where to resume after a batch is processed, or if our system crushes. The match result data is stored into MongoDB and the information needed to download game replays is delivered to the "Data Extraction" part. Here the replays are downloaded, parsed and stored into MongoDB. Neo4j loader extract data from MongoDB and store it into Neo4j.

After the ETL, we use Spark as the analytics engine to answer our business questions. The results are visualized by Tableau. For the relationship BQ, we use the built-in Neo4j Browser to visualize the relationship between professional teams.

4 Data Analysis

We use Spark as our analytics engine to analyze business questions raised in our proposal. We don't fulfill all business questions and we will state the reason for it in the challenges section. We have 3 game types: the public game, the ranked game, the professional game. Most following business questions are answered in these three game types. So most of them applied filtering games by their types.

4.1 Bad Manner Hero

This analysis is aimed to find out the hero which has the most possibility to be a bad manner player.

We define a bad manner as leaving a game before the game ends. We use match result data, since it contains leave game status. We do aggregation on the hero id to get the pairs of a hero id and a leave game count. Before further steps, we first check if there is no bad manner player, which turns out to be true in professional games. Then for each pair, we divide the leave game count by the total pick count of this hero to compute the possibility. Finally, we can get the bad manner hero by sorting the processed pairs. In public games, hero which has the most possibility to be a bad manner player is Broodmother, who has 10% leave game rate. In ranked games, it is Meepo, who has 5% leave rate. And in professional games, there are almost no players leave a game before a game ends. Through the result, we can find out that players are more serious in ranked games than the public, since ranked games affect their rankings.

4.2 Game Time

This analysis is aimed to find out the average time of a game. We use the match result data which has the game duration. For each match result data, we can extract the duration and increment the total game count by 1. Finally, we can get the average cost of time by dividing the sum of all duration by the total game count. For public games, the average time is 33 minutes. For ranked games, the average time is 40 minutes. For professional game, the average time is 30 minutes.

4.3 Damage/Gold Rate

This analysis is aimed to find out the heroes who have the top 20 damage rate and the heroes who have the bottom 20 damage/gold rate.

We define the such rate of a hero as "the damage to other heroes per gold gained by the hero". We use match result data which contains damage to other heroes, gold spent, and rest gold of each hero. For each match result, we extract the 3tuple of a hero id, a total damage to other heroes, a total gained gold (addition of gold spend and rest gold). Then we do an aggregation on the hero id and divide the damage by the gained gold to get pairs of a hero id and a damage/gold rate. Finally, to get the top 20 heroes and the bottom 20 heroes, we can simply sort pairs by the damage rate.

The figure in the next page demonstrates the result. A table for this can also be found in the Appendix.

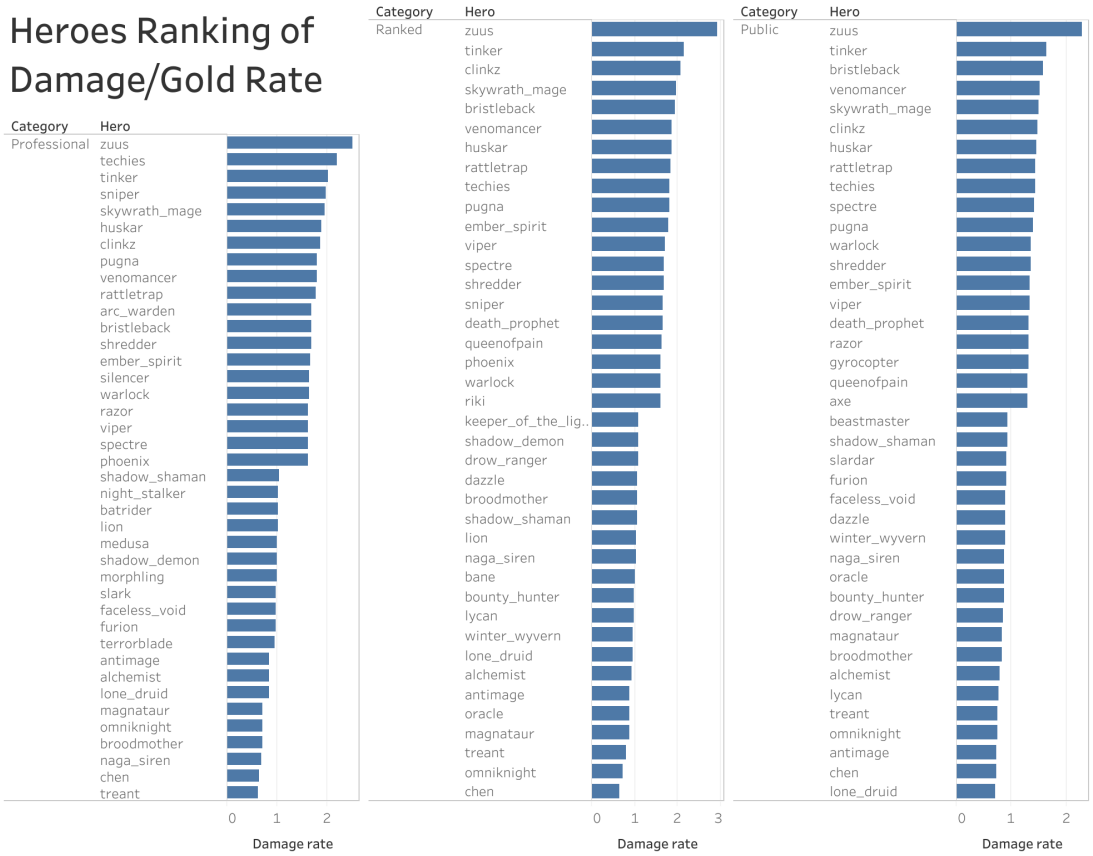

Figure 3: Heroes Ranking of Damage/Gold Rate

4.4 Economy Distribution vs Game Result

This analysis is aimed to find out whether there exists a correlation between economy distribution in a game (evenly distributed or centralized) and the game result.

We define the economy as gold gained in a game It's possible that even distribution is better and it's also possible that having good players gaining more gold is better.

To examine the hypothesis, we applied logistic regression. We use the match result data which contains the amount of gold gained per minute for each player. For each match result data, we extract a pair of a game result and a normalized gold distribution. A gold distribution contains the proportion of gold of 5 players in a game. For each distribution, we sort it in ascending order to make the smallest proportion at the head of the distribution. Then we can use sorted distribution as the data point for the logistic regression model where each proportion of gold is a feature. The game result is used as the label. After running the training data, we find the accuracy is only 0.53 which is close to making a guess. Thus, our hypothesis turns out to be false.

4.5 Gain in First 15 Minutes

This analysis is aimed to find out the hero who gains most XP or gold in the first 15 minutes of the game.

We use replay data which contains all events during a game. We firstly filter all events irrelavant of XP or gold, and secondly extract the time of each event happened and filter it. Then we can do aggregation on hero name to calculate the total gain XP/gold. Finally, we sort the gained XP/gold value to compute the hero who gains most XP/gold in 3 game types. We list the result for this analysis in the Appendix.

4.6 First Blood Time vs Game Time

This analysis is aimed to find out the correlation between the time of first blood and the total cost of time of a game. The first blood is the first kill in a game. We guess early first blood implies a short game.

To examine our guess, we apply linear regression in this analysis. We use match result data which contains the first blood time and the total duration. For each match result, we extract the first blood time and the game result. The data point used in linear regression is the time of the first blood. The label used is the game result. But the root mean squared error on training data is 765 which is large. The error shows that there is no correlation between the first blood time and the cost of time.

4.7 Hero Statistics

This analysis is aimed to find out the top 5 heroes who have most kills/assists/deaths/heals.

We use replay data which contains all events during a game. We firstly filter by the event type we want (kill/assist/death/heal). For each replay data, we extract a pair of a hero id and a count of the event type. Then we do aggregation on the hero id and sort pairs by the count to extract the top 5 heroes.

The following figure demonstrates the result in professional games. Other results are listed in the Appendix.

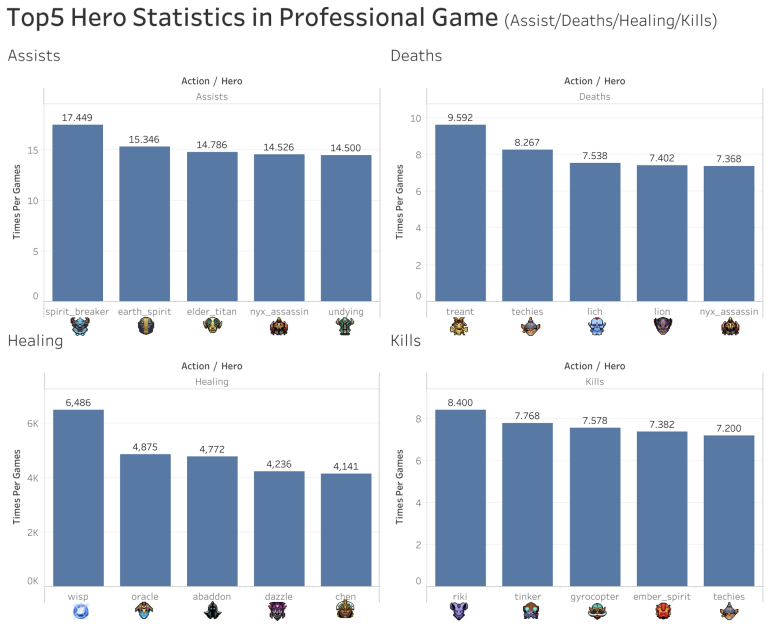

Figure 4: Top 5 Hero Statistics in Professional Game

4.8 Most Ban

This analysis is aimed to find out the top 5 heroes who are banned in professional games (only professional game allows banning heroes).

We use match result data which contains banned heroes. We first filter games to keep only professional games and then extract a pair of a hero id and 1 (denotes banned once) for each match result data. After doing aggregation on the hero id, we sort the banned count to get the top 5 banned heroes. They are Alchemist, Enchantress, Shadow Demon, Faceless Void, and Life Stealer. And we can find some of them has a good ranking in the previous analysis. For example, Alchemist has a 7th damage/gold rate in public games.

4.9 Most Pick

This analysis is aimed to find out the top 5 heroes who are picked.

We use match result data which contains picked heroes of each game. We extract pairs of a hero id and 1 (denotes picked once). After doing aggregation on the hero id, we sort the picked count to get the top 5 picked heroes.

The figure in the next page demonstrates the result of the above two questions. A table for them can also be found in the Appendix.

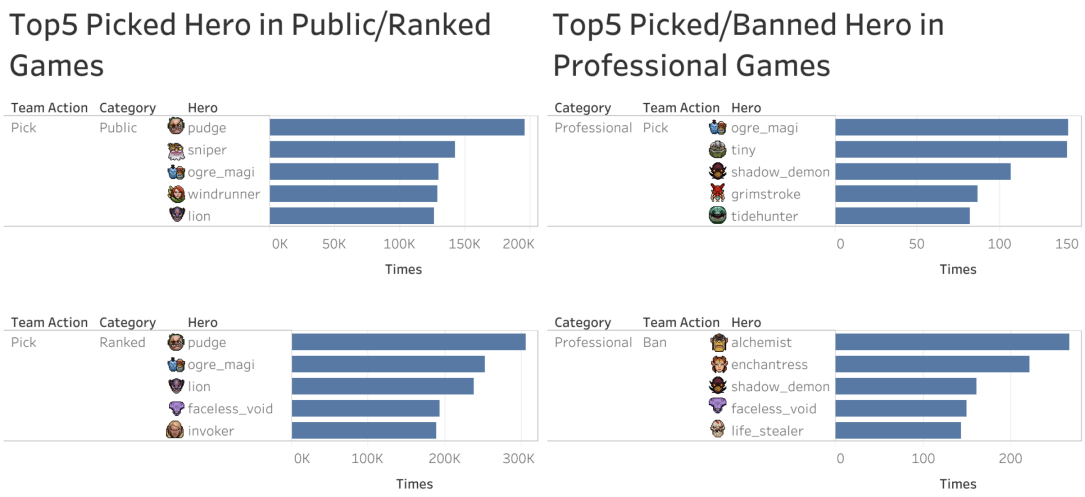

Figure 5: Top 5 Picked/Banned heroes

4.10 Most Used/Purchased Item

These two analysis are aimed to find out the top 5 most used/purchased items.

We use replay data in this analysis because it contains all events happen during a game. In Most Used Item analysis, we filter events contained in replay data to get all 'use item' events. Then we do aggregation on the item id to get the pairs of an item id and an item use count. Finally, we can get the top 5 most used items by sorting the pairs by the use count. We have similar steps in Most Purchased Item analysis.

The figure in the next page demonstrates the result. A table for this can also be found in the Appendix.

Figure 6: Top 5 Used/Purchased Items

4.11 Team Battle Detector

This analysis aimed to find out the average first time of a team battle and the average times of team battle happens in public, ranked, and professional games.

We use the replay data which has all events. But these events do not contain the location information. Thus, we roughly define the team battle as in 3 minutes there are 4 or more hero deaths. A sliding-window algorithm is applied to build a team battle detector. We extract the death events and use the detector to detect all team battles in a game. The time of the first team battle is at 11 minutes, 13 minutes and 19 minutes in public, ranked, and professional game. The average number of team battles is 7, 9, and 6 in public, ranked, and professional game respectively.

Additionally, for most analysis above, we can only get the hero id or item id. To provide better readability, we add two metadata collection heros and items which contain hero/item ids and their names. So after getting the id, we do one more query on the matadata collections to get the its name.

4.12 Win-lose Relationship Visualization

This analysis aimed to visualize the result between the professional teams. It could serve as a proof of concept that we have the ability to further explore other relational topics.

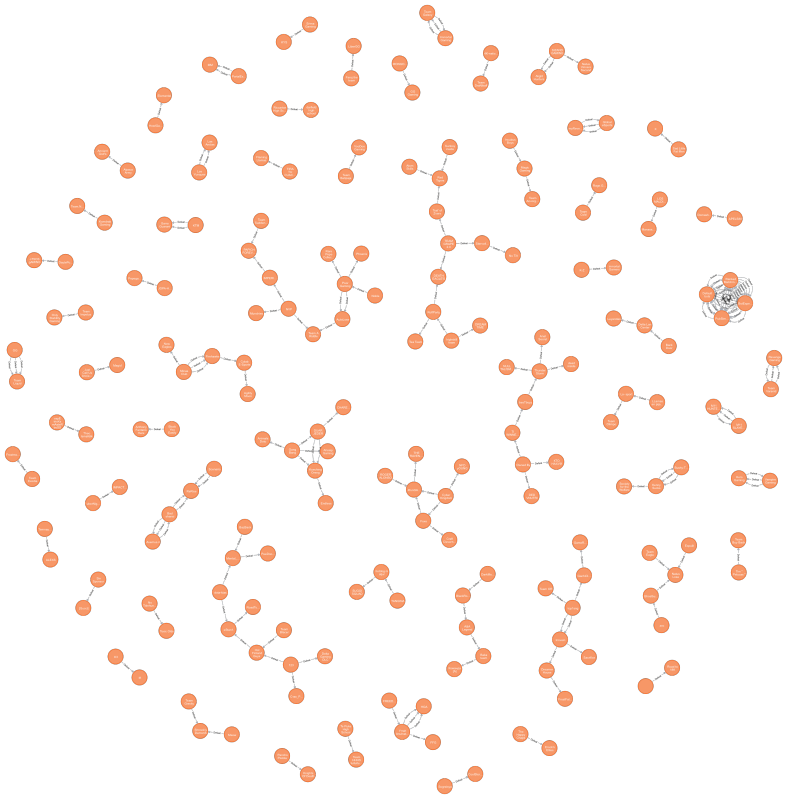

Figure 7: Win-lose Relationship Visualization Result

5 Challenges

5.1 Valve API

Valve provides us with various apis to get match results, but they're not well documented and changes all the time without providing changelogs.

At the beginning we tried to use GetMatchHistory API, which seems to be very straightforward in the documentation. It says we can specify a starting match ID and will return a bunch of match results starting with it. Unfortunately, specify such an ID is actually disallowed and it can just return the newest results that are currently produced, and this was not documented.

We have also thought the GetMatchHistory API can be used to make a data stream for stream analysis, but this couldn't be done to since Valve limited its api calling gap. It's said to be 100,000 calls per day but actually you can do much less. I set the api calling interval to 6 seconds and sometimes it still denies the calls. Considering the velocity that the game results are produced we can't make such a stream to catch all the game results that are being produced.

After exploring the Dota2 developers' forum for a while, we found out that we can use the GetMatchHistoryBySequenceNum API, which works similarly as the before one but this time we need to specify a starting sequence number, which is also a unique identifier for each game, but we can't see it in the Dota2's client. This problem is solved by finding out that the sequence number can be found in the GetMatchDetails API. We can first choose

a starting sequence number by looking for a match whose date is around the starting date of all the matches er want to collect. By calling GetMatchHistoryBySequenceNum, we can get a list of match results, then the sequence number of the last match among the results can be extracted. Adding one to it will serve as starting sequence number of the next request. By doing this repeatedly, we can collect as many matches we want.

As I mentioned, the api call limits are much less than it is described and it seems to be random. I created a request key pool to randomly select a key for the api call each time and if it was denied, we waited for 3 seconds then do the next call.

5.2 Dota2 replays Parameters

The request to download Dota2 replays requires three parameters, `cluster` , `matchid` , and `replaysalt` . In the past, by calling GetMatchDetails API with match ID of a game, the cluster number and "replay salt" of a specific match can be acquired. However, now Valve disallowed us to do so, by hiding the "replay salt" field it the result.

Fortunately, we found another https://docs.opendota.com/ that can help finding the fields needed to download a replay.

Furthermore, we thought of solely rely on it to get the data in the first dataset, the match results. In this way, only one kind of api is used which can simplify the data pipeline. But this can't be done neither, this third party api will ask us to pay if our request exceeds a certain number, that will cause cost exceeding our budget (which is zero). Also, we don't want to collect all the replays. It's clear that the matches played by the professional teams worth more than the normal ones. Given the limited time and storage we have, we decided to first collect a certain number of professional games, then collect public games and ranked games of the same number for comparison.

5.3 Docker

As mentioned in the proposal, we tried to containerized the project by docker, but it ends up with failures. Considering to the limited time, we decided to discard using docker for installing our NoSQL technology.

At first, I tried to dockerize our project by using a single Dockerfile. We used ubuntu as a parent image so that docker can virtualize an ubuntu operation system environment on our host machine, and then install needed packages in it. However, we can not connect to the database. Realizing that it is cramped to put all the services on a Dockerfile, we tried to construct a docker-compose.yml to define, run, and scale services with Docker.

According to our data pipeline, I intended to build 4 services - FetchStore, ParseReplay, MongoDB, and Analytics - in the docker-compose. In the docker-compose, we can also use the Dockerfile by using the command `build` in a service. Therefore, I constructed Dockerfiles for FetchStore, ParseReplay, and Analytics services. And I used the official image of MongoDB, and used `volume` in the service to reflect the direction of configuration file and MongoDB initial JavaScipt. Also, I utilized `depends_on` to define simple interaction between the services. There are a lot of things to consider when running multiple containers, and it's easy to make mistakes if not being thoughtful enough.

As a result, we still had a hard time connecting to the database. When this problem was fixed, we only finished the configuration of the FetchStore and MongoDB service. Then we realized we got limited time left. Therefore, we discard docker for now and move on to deploy our project on amazon web service (AWS). And we hope to complete this part in the future.

5.4 Setup AWS environment

Because we planned to use over 200GB data, we decided to use AWS EC2 to run our system. We chose EBS (Elastic Block Store) which we thought would automatically have larger storage when our instance is full. But

during fetching data, we still met the error which said the file system is full. Then we manually added more storage volume to solve this problem.

5.5 Result Readability

We have 2 types of data. But in match result data, the heroes and items inside are denoted by their id. But in replay data, they are denoted by their name. Obviously, using a name can increase readability. Thus, we added 2 more metadata collections to store the map from id to name.

5.6 Corrupted Data

Although we did some data validation before using it, there are still some unexpected corrupted data problems. For example, one replay data loses part of it which makes out result bad. So we added more filters in analysis, like the filter to the duration of a game.

5.7 Setup Neo4j on AWS

We want to setup Neo4j server on AWS and can be connected by a Neo4j browser on a local laptop. But the find the connection was failed. We first modified the configuration file of Neo4j which doesn't allow remote access in default, which, does not work out. Then we found out that we should modify the security group setting on AWS which controls access to the AWS instance.

5.8 Missing Output on Console

Firstly, we just printed the result on the console. But after we deployed our system on AWS, we found a problem that the connection from local machine to AWS might be broken. If so, we cannot get the result printed on the console. As a result, besides printing result on the console, we also write the result to a file, which prevents missing the output on the console.

5.9 Limited Information in Parsed Replay Data

As mentioned in Analysis, some business questions in proposal are not fulfilled because we find that the parsed replay data doesn't contains the information we need. One important information is location. Some of our business questions rely on the location information, for example, analysis on the path of heroes and the lane. Actually, this problem is caused by the parser we used because a replay data contains almost all information of a game. We known this fact when we wrote the proposal, but we still propose these questions. Because we though we can try to find another parser which can extract the location information from the replay data. But we didn't find one which makes our give up these questions.

6 Appendix

6.1 Detailed Data Dictionaries

6.1.1 Match Result Dataset

Field Name	Data Type	Description	Example
players	JSON Array	Status about each player at the end of the game, see table 3	See table 3.

Field Name	Data Type	Description	Example
radiant_win	boolean	Dictates the winner of the match, true for radiant; false for dire.	true
duration	int	The length of the match, in seconds since the match began.	2452
start_time	long	Unix timestamp of when the match began.	1566731204
match_id	long	The matches unique ID.	4986382845
match_seq_num	long	A 'sequence number', representing the order in which matches were recorded.	4182489897
tower_status_radiant and tower_status_dire	int	See API doc for details.	260
barracks_status_radiant and barracks_status_dire	int	See API doc for details.	51
first_blood_time	int	The time in seconds since the match began when first-blood occurred.	102
lobby_type	int	The type of the match	0
human_players	int	The amount of human players within the match.	10
leagueid	int	The league that this match was a part of. A list of league IDs can be found via the GetLeagueListing method.	0
positive_votes	int	The number of thumbs-up the game has received by users.	0
negative_votes	int	The number of thumbs-down the game has received by users.	1
game_mode	int	The mode of the match	11
picks_bans	JSON Array	A list of the picks and bans in the match, if the game mode is Captains Mode.	Details will be described in another table
radiant_score	int	The total amount of kills that radiant team make	52
dire_score	int	The total amount of kills that dire team make	42

Table 2: Match Result Data Dictionary

The data dictionary for one JSON object of field ”players” in table 2 is as follows.

Field Name	Data Type	Description	Example
account_id	long	An unique identifier for each steam account	4294967295
player_slot	int	see this in details https://wiki.teamfortress.com/wiki/WebAPI/GetMatchDetails#Player_Slot	1
hero_id	int	The hero’s unique ID. A list of hero IDs can be found via the GetHeroes method.	97
item_0 to item_5	int	item id of the inventory item	48
kills	int	The amount of kills attributed to this player.	10
deaths	int	The amount of times this player died during the match.	2
assists	int	The amount of assists attributed to this player.	7
leaver_status	int	Indicating whether a player disconnect from the game before the game ends	0
gold	int	The amount of gold the player had remaining at the end of the match.	32341
last_hits	int	The amount of last-hits the player got during the match.	109
denies	int	The amount of denies the player got during the match.	24
gold_per_min	int	The player’s overall gold/minute.	601
xp_per_min	int	The player’s overall experience/minute.	587
gold_spent	int	The amount of gold the player spent during the match.	21239
hero_damage	int	The amount of damage the player dealt to heroes.	34034
tower_damage	int	The amount of damage the player dealt to towers.	8902
hero_healing	int	The amount of health the player had healed on heroes.	452
level	int	The player’s level at match end.	23

Table 3: Player Field in Match Result

The data dictionary for one JSON object of field ”picks_bans” in table 2 is as follows.

Field Name	Data Type	Description	Example
is_pick	boolean	Whether this entry is a pick (true) or a ban (false).	true
hero_id	int	The hero’s unique ID.	20
team	int	The team who chose the pick or ban; 0 for Radiant, 1 for Dire.	1
order	int	The order of which the picks and bans were selected.	10

Table 4: Ban_pick Field in Match Result

6.1.2 Parsed Replay Dataset

The data dictionary for one JSON object of field ”combatlog” in table 1 is as follows.

Field Name	Data Type	Description	Example
time	double	The time when this action happens	325.608734130432
type	String	The type of this action.	”purchase”
target	String	The target of this action	”npc_dota_hero_nevermore”
item	String	This field only occurs when the type of the action is ”purchase”. Indicates which item was bought by this action	”item_enchanted_mongo”
inflictor	String	This field only occurs when the type of the action is ”damage”, ”lose_buff” and ”add_buff”. The inflictor of the current action	”modifier_tower_aura_bonus”
damage	int	This field only occurs when the type of the action is ”damage”, which indicates the damage caused by the current action.	80
before_hp	int	This field only occurs when the type of the action is ”damage”, which indicates the hp of the target before the action.	725
after_hp	int	This field only occurs when the type of the action is ”damage”, which indicates the hp of the target after the action.	718

Table 5: Combatlog Field in Replay

The data dictionary for one JSON object of field ”info” in table 1 is as follows.

Field Name	Data Type	Description	Example
game_winner	int	1 if radiant wins and 2 if dire wins.	1
leagueid	int	The unique league id of the current professional game, 0 if it is not a professional match.	4122
match_id	long	The unique identifier for each game.	4986514901.
end_time	long	The timestamp of the ending of the game.	1566736571.
game_mode	int	The mode of the current game	21

Table 6: Info Field in Replay

6.2 Detailed Analysis Results

6.2.1 Damage/Gold Rate

Public Games:

Rank	Hero	Damage/Gold Rate
1	Zuus	2.3004157401846537
2	Tinker	1.6420031038927647
3	Bristleback	1.5771368121914766
4	Venomancer	1.5322559346116191
5	Skywrath_mage	1.5017875157064025
6	Clinkz	1.4849750590792063
7	Huskar	1.4546674116635778
8	Rattletrap	1.439648234092895
9	Techies	1.4383042472835008
10	Spectre	1.42678633556202
11	Pugna	1.3972072332418493
12	Warlock	1.3632018454946753
13	Shredder	1.35394499161979
14	Ember_spirit	1.3475096725161406
15	Viper	1.3412099350143785
16	Death_prophet	1.3278598501932295
17	Razor	1.3192034122801117

Rank	Hero	Damage/Gold Rate
18	Gyrocopter	1.3112357419375424
19	Queenofpain	1.3066481175265756
20	Axe	1.2931725267257896

Table 7: Public Games Damage/Gold Rate

Ranked Games:

Rank	Hero	Damage/Gold Rate
1	Zuus	2.9418056479438626
2	Tinker	2.150859003299626
3	Clinkz	2.0884421511990525
4	Skywrath_mage	1.9760465643357101
5	Bristleback	1.9501640391382127
6	Venomancer	1.881336238794785
7	Huskar	1.8699499986263792
8	Rattletrap	1.8323486050421012
9	Techies	1.822758345306855
10	Pugna	1.818287376158369
11	Ember_spirit	1.7799193703114184
12	Viper	1.713903523823672
13	Spectre	1.696238851802073
14	Shredder	1.677920019400073
15	Sniper	1.651809735988255
16	Death_prophet	1.6508798569324052
17	Queenofpain	1.633404234350671
18	Phoenix	1.6167379442865089
19	Warlock	1.6103239765043673
20	Riki	1.6019336243768927

Table 8: Ranked Games Damage/Gold Rate

Professional Games:

Rank	Hero	Damage/Gold Rate
1	Zuus	2.5095296155046065
2	Techies	2.1870154722684325
3	Tinker	2.0124433616059756
4	Sniper	1.9783706295917614
5	Skywrath_mage	1.9535457462792696
6	Huskar	1.8735118229824992
7	Clinkz	1.8606849160752506
8	Pugna	1.7964929829557952
9	Venomancer	1.7848610595534815
10	Rattletrap	1.7661035883036698
11	Arc_warden	1.6768414694470062
12	Bristleback	1.6745098255685582
13	Shredder	1.673216393476275
14	Ember_spirit	1.6568609854739513
15	Silencer	1.6489648349788801
16	Warlock	1.6457797988727485
17	Razor	1.613006088280061
18	Viper	1.6119751502444593
19	Spectre	1.611136567135945
20	Phoenix	1.610776975596254

Table 9: Professional Games Damage/Gold Rate

6.2.2 Hero Statistics

Public Games:

Rank	Type	Hero	Average
1	Kill	Skywrath_mage	9.669361281499235
2	Kill	Riki	8.918664447370382
3	Kill	Bloodseeker	8.867892841571528
4	Kill	Huskar	8.842363960058828
5	Kill	Clinkz	8.620800198367014

Rank	Type	Hero	Average
1	Assist	Spirit_breaker	17.035058859815965
2	Assist	Zuus	16.959805989951498
3	Assist	Disruptor	16.16634037197595
4	Assist	Undying	16.01732058393167
5	Assist	Ancient_apparition	15.772262124197546
1	Death	Shadow_shaman	8.704649170353015
2	Death	Huskar	8.60463271151018
3	Death	Earth_spirit	8.539345711759506
4	Death	Venomancer	8.261024100529708
5	Death	Crystal_maiden	7.996521181001284
1	Healing	Wisp	8113.96989272119
2	Healing	Dazzle	6512.851949097656
3	Healing	Oracle	4927.196047300997
4	Healing	Chen	4643.917551963049
5	Healing	Juggernaut	4531.026821805381

Table 10: Top Hero Statistics for Public Games

Ranked Games:

Rank	Type	Hero	Average
1	Kill	Meepo	12.175918939312277
2	Kill	Storm_spirit	11.574019337016574
3	Kill	Skywrath_mage	11.382918886915334
4	Kill	Ursa	11.381844281905803
5	Kill	Clinkz	11.360121362999845
1	Assist	Spirit_breaker	21.138613691680092
2	Assist	Zuus	19.846135935482504
3	Assist	Undying	19.50708614360113
4	Assist	Disruptor	19.030801114843136
5	Assist	Ancient_apparition	18.42858021577866
1	Death	Shadow_shaman	10.57965933174599

Rank	Type	Hero	Average
2	Death	Earth_spirit	9.929016097543007
3	Death	Venomancer	9.770501670227812
4	Death	Witch_doctor	9.737475719383838
5	Death	Crystal_maiden	9.699951072901378
1	Healing	Wisp	9000.57524051552
2	Healing	Dazzle	7985.429513362119
3	Healing	Chen	6758.288667205896
4	Healing	Oracle	6579.915818722605
5	Healing	Juggernaut	6238.096888357454

Table 11: Top Hero Statistics for Ranked Games

Professional Games:

Rank	Type	Hero	Average
1	Kill	Riki	8.4
2	Kill	Tinker	7.767857142857143
3	Kill	Gyrocopter	7.578313253012048
4	Kill	Ember_spirit	7.381818181818182
5	Kill	Techies	7.2
1	Assist	Spirit_breaker	17.448979591836736
2	Assist	Earth_spirit	15.346153846153847
3	Assist	Elder_titan	14.785714285714286
4	Assist	Nyx_assassin	14.526315789473685
5	Assist	Undying	14.5
1	Death	Treant	9.591836734693878
2	Death	Techies	8.266666666666667
3	Death	Lich	7.538461538461538
4	Death	Lion	7.402298850574713
5	Death	Nyx_assassin	7.368421052631579
1	Healing	Wisp	6486.472222222223
2	Healing	Oracle	4874.9375

Rank	Type	Hero	Average
3	Healing	Abaddon	4772.2
4	Healing	Dazzle	4236.021739130435
5	Healing	Chen	4141.310344827586

Table 12: Top Hero Statistics for Professional Games

6.2.3 First 15 min Gold/XP Gain

Game Type	XP or Gold	Hero
Ranked	XP	Broodmother
Ranked	Gold	Broodmother
Public	XP	Meepo
Public	Gold	Arc_warden
Professional	XP	Visage
Professional	Gold	Broodmother

Table 13: First 15 min Gold/XP Gain

6.2.4 Most Pick Hero

Public Games:

Rank	Hero	Time
1	Pudge	195833
2	Sniper	142191
3	Ogre_magi	129292
4	Windrunner	128924
5	Lion	126034

Table 14: Most Pick Hero for Public Games

Ranked Games:

Rank	Hero	Time
1	Pudge	306246
2	Ogre_magi	252735

Rank	Hero	Time
3	Lion	238215
4	Faceless_void	193762
5	Invoker	188404

Table 15: Most Pick Hero for Ranked Games

Profressional Games:

Rank	Hero	Time
1	Ogre_magi	142
2	Tiny	141
3	Shadow_demon	107
4	Grimstroke	87
5	Tidehunter	82

Table 16: Most Pick Hero for Professional Games

6.2.5 Most Purchased/Used Item

Public Games:

Rank	Purchased/Used	Item	Total Times
1	Purchased	Tpscroll	83564
2	Purchased	Ward_sentry	31378
3	Purchased	Ward_dispenser	25616
4	Purchased	Ward_observer	23507
5	Purchased	Boots	11906
1	Used	Phase_boots	202182
2	Used	Tpscroll	79941
3	Used	Bottle	57859
4	Used	Blink	52500
5	Used	Ward_dispenser	41044

Table 17: Most Purcahsed/Used Item for Public Games

Ranked Games:

Rank	Purchased/Used	Item	Total Times
1	Purchased	Tpscroll	99078
2	Purchased	Ward_sentry	48867
3	Purchased	Ward_dispenser	44732
4	Purchased	Ward_observer	35551
5	Purchased	Clarity	15273
1	Used	Phase_boots	315978
2	Used	Tpscroll	95651
3	Used	Bottle	87434
4	Used	Blink	75464
5	Used	Ward_dispenser	71248

Table 18: Most Purcahsed/Used Item for Ranked Games

Professional Games:

Rank	Purchased/Used	Item	Total Times
1	Purchased	Tpscroll	28448
2	Purchased	Ward_sentry	12195
3	Purchased	Ward_dispenser	11834
4	Purchased	Ward_observer	11361
5	Purchased	Clarity	6383
1	Used	Phase_boots	109046
2	Used	Bottle	32853
3	Used	Tpscroll	27472
4	Used	Power_treads	25600

Table 19: Most Purcahsed/Used Item for Professional Games